Improving the Cycle Lengths of Chaotic PRNGs

Gabriele Spenger, Jörg Keller
Faculty of Mathematics and Computer Science
FernUniversität in Hagen
58084 Hagen, Germany

Abstract

Chaotic functions have been announced in the literature as promising for implementing low complexity pseudo-random number generators (PRNGs) required e.g. for RFID security applications. They combine good theoretical statistical properties with a computationally simple algorithm. Unfortunately, actual implementations with finite number precision show a disappointing behavior compared to the mathematical theory. This results for example in comparably short cycles in the state space graph, which lead to a repetition of the generated pseudo random values after a small number of iterations. This paper presents a simple way to improve the state space structure of chaotic PRNGs by using a different parametrization of the chaotic function at certain iterations and hereby breaking out of these cycles. This approach reduces this aspect of the weakness of such implementations, which we demonstrate with several examples. The statistical properties of the modified algorithms are evaluated with the NIST test battery.

1. Introduction

The growth of electronic communication over the last decades and the development of technologies like Radio Frequency Identification (RFID) (or more general the Internet of Things) has led to a large interest in data security in all kinds of devices. Sending sensitive information over communication channels that are accessible by attackers, e.g. the Internet, or the air in case of radio transmission, requires measures to secure the confidentiality as well as the integrity of the transmitted data. In order to achieve this, cryptographic protocols have been developed and standardized that make use of cryptographic base functions like symmetric or asymmetric encryption, hashing and pseudo random number generators (PRNGs). Because of the limitations in cost, energy consumption and computational performance for devices like RFID transponders, low complexity cryptographic functions are of high interest for applications running on these devices (cf. categories by Manifavas et al [14]).

Chaotic functions have shown promise for providing low complexity PRNGs because they combine good theoretical statistical properties with a computationally simple algorithm [19]. Unfortunately, actual behavior of chaotic PRNGs is often disappointing in terms of period length due to limited precision number representations [11]. While increasing the precision helps to overcome this, it puts a burden on the hardware or the computational load and thus is not always an option. Another proposal by Mennink and Preneel [16] cures the problem by combining the outputs of two independent PRNGs. Yet, this also doubles hardware resource or computational load.

We present a novel solution that breaks out of short cycles by applying a second transition function from time to time. Because the other function can be a variant (only differing in some parameter settings) of the normal transition function, hardware and/or computational load is only marginally increased. We analyze our approach theoretically and show that it increases expected cycle lengths towards results expected for truly random functions. We evaluate our proposal in several settings and demonstrate encouraging results.

The remainder of this paper is organized as follows. Section 2 provides basic information about Pseudo Random Generators as well as the analysis and desired properties of the related state space graph. Section 3 introduces chaotic functions and presents analysis results of the state space. Section 4 describes the approach of breaking out of the cycles in detail and presents results for a random approach to the target node selection. Section 5 enhances the approach and compares the analysis results. Section 6 presents a statistical analysis of the results using the NIST test battery. Section 7 concludes the paper and provides an outlook to potential future investigations.

2. Basics

A couple of methods have been standardized for the assessment of the suitability of cryptographic functions. Examples for such methods targeted to PRNGs are the Marsaglia suite of Tests of Randomness 0, the Entacher collection of selected pseudorandom number generators [6] and the NIST test suite [17]. While these test batteries provide valuable information about the statistical properties of the output from a given PRNG, they do not examine the complete state space of the algorithms. Instead,

their analysis is based on the statistical properties of a limited number of output values. Algorithms that do not meet the pass criteria of the test batteries are usually not suited for security applications, because the statistical properties might be exploited by an attacker to predict future generated output values. But algorithms that do pass the criteria do not necessarily provide high security. This fact is typically included as advice in the documentation of the test batteries, e.g. in the abstract of the NIST publication. Further examples for non-statistical properties of PRNGs that are suitable for cryptographic purposes include Forward Secrecy and Backward Secrecy (high difficulty to calculate past or future generated outputs or states from a compromised current random number) [15]. Still, if the cycle length of a PRNG is short, patterns of output bits can be stored and repetition detected. A practical example for this is the attack on A5/1 (cf. [3], [8]). Thus, a long cycle length is a prerequisite to enable forward secrecy and further security models for PRNGs e.g. [4] and [5].

Pseudo Random Number Generators are generally deterministic state transition functions $f: M \to M$ mapping a finite state space of size $n = |M|$ to itself if they do not receive new seed or entropy bits. Every output of the PRNG results in a state transition. This means that the generated sequences of pseudo random numbers are periodic. The output is deterministic and dependent on the state. Therefore, only the state is considered in the following. If a single state is interpreted as a node and the transition between a state and its unique successor state is interpreted as an edge, the result is a directed graph $G_f = (V; E)$ with $V := M$ and $E := \{f(x; f(x)\} | x \in M\}$. The structure of the generated graph provides information about the behavior of the pseudo random generator. For non-bijective transition functions, the graph typically consists of several weakly connected components. Each of these components consists of one cycle and generally several trees with roots located on the cycle. Figure 1 depicts the structure of a component.

Properties of the graph include the number and sizes of the connected components, length of the cycles and maximum depth of the trees. In order to identify all connected components of a graph, the complete state space would have to be analyzed, e.g. by a depth first search. The cycles can be detected by starting from the unique back edge in each component. Alternatively, only a part of the state space can be analyzed, accepting the fact that one or several components might be missed. Still, this approach can provide valuable information about the expected state space structure.

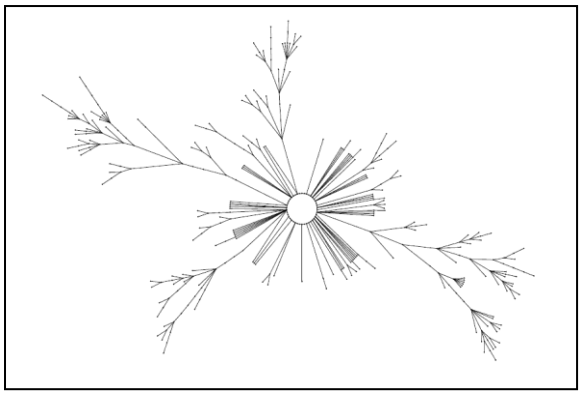

Figure 1. A typical connected component of a state transition graph [2]

2.1 Sampling the state space

Obviously, the approach of sampling the state space has the risk of missing a weakness of the state graph, as the sampling results in only a very small part of the state space being analyzed. For this reason, this approach is not suitable for positively approving the security of an algorithm. Instead it can be used to randomly find weaknesses such as short cycles, which might be sufficient to make an algorithm unsuited for a given use case. If random sampling of the state space shows a weakness, chances might be good that more instances of the same weakness exist in the state space.

The impact of such a weakness is even higher than obvious on first sight: if the number of components is small (as expected, cf. Section 2.2), then some component must be large, and already a small number of sample paths through the state graph will provide the cycle lengths of the larger components. This weakness accordingly affects a majority of starting points in the state graph.

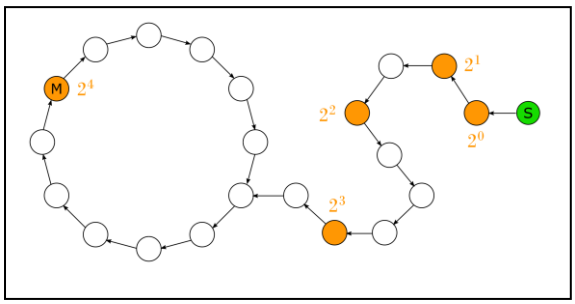

Figure 2. A cycle is detected

In [10] an analysis method of the state space is presented that avoids storing a component number for every node (as DFS does). The idea is to only store certain nodes while traversing the tree. If only the

nodes are stored that are reached after $2, 2^2, 2^3, \ldots$ steps taken since the start value, the required memory usage is logarithmically in the number of steps, i.e. $O(\log n)$. This results in a low memory requirement even for extremely long runs. Still, the cycle at the end of a path can be detected as ultimately one of the stored nodes is reached again, and the cycle length can be computed. Figure 2 illustrates the process of cycle detection.

This method allows performing an analysis of a large state space. The memory required for the analysis is only logarithmically growing with the size of the state space. Still, the required time for the analysis prevents to examine the complete graph for $n \gg 2^{40}$, which also seems the border for DFS (even with external memory algorithms). But it allows taking a number of randomly chosen start values and walking through the graph starting from these, as long as the length of a path is smaller than about 2^{50} (which restricts n to 2^{100}, see next subsection). This way, the state space is sampled and the resulting tree and cycle structure for these samples might provide valuable insights with respect to the security of the algorithm. Also, the sizes of the connected components can be guessed from the sampling within a confidence interval.

Please note that there are comparable approaches for bijective functions, notably Knuth's algorithm [12] with an expected runtime of $O(n \log n)$, which can be made linear in time by using 1 bit per node.

2.2 Metrics for a "good" PRNG

While information about the structure of a state transition tree provides valuable insight about the properties of a PRNG, this information needs to be compared to the structure of a "good" PRNG. Obviously, the meaning of "good" and "bad" (and a clear separation between the two classes) depends heavily on the application and thus on the security model. Therefore, the definition of such criteria is out of the scope of this document (cf. e.g. [4] and [5]), but some reasonable guidance is provided in the following. The properties that can easily be deducted from the structure analysis are:

- Number of components (at least of larger components)
- Cycle lengths of the components
- (Relative) Sizes of the components
- Tree heights of the components

It is desirable that a PRNG has a small number of components. The fewer components are present in the state structure, the larger can the components and their cycle lengths be. According to [7], the expected number of components for a randomly chosen non-bijective state transition function on a set M with n elements is $0.5 \cdot \log n$. For an invertible state transition function, the expected value is $\ln n$ [18]. For a "good" PRNG this value should not be less than the expected value.

Furthermore, it is desirable that the number of nodes on cycles (which is equivalent to the sum of cycle lengths) is as large as possible. This results in a high number of steps until the states are repeated. According to [7], the expected sum of cycle lengths for non-bijective state transition functions is $\sqrt{\pi n/2}$. The largest cycle length is expected to be $0.78 \cdot \sqrt{n}$. A "good" PRNG should have one cycle with at least a comparable size. For bijective state transition functions the expected length of the largest cycle is $\left(1 - \frac{1}{e}\right) \cdot n = 0.632 \cdot n$.

The number of nodes in any component (equivalent to the size of the component) should be as large as possible. Ideally the state structure consists of a single component containing all nodes of the graph. According to [7], the expected number of nodes of the largest component for a non-bijective state transition function is about $0.76 \cdot n$. PRNGs with bijective state transition functions consist of cycles only and therefore the component sizes equal the cycle lengths.

Finally, the largest tree consists of about $0.5n$ nodes with a depth of about $\sqrt{n}$ [7], i.e. it comprises about $2/3$ of the nodes in the largest component. As the trees in the considered graphs are ragged and the nodes are distributed rather symmetrically over the depths of the tree, the average tree path length from the start node to the entry into the cycle observed should be at least half of the tree height, i.e. $2/3 \cdot 1/2 \cdot \sqrt{n} = 0.33 \cdot \sqrt{n}$ (cf. [7]).

3. Chaotic functions

The definitions of a chaotic function $f: R \to R$, where R is an interval in the reals, vary widely. Most definitions agree on the following properties of a sequence of points $x, f(x), f(f(x)), \ldots$ (cf. [11]):

- f reacts sensibly to changes in x, i.e. even small changes to the value of x result in large changes of the sequence.
- f is topologically transitive, i.e. almost every element of R can be connected to almost every other element of R by a finite sequence.
- f is topologically dense, i.e. even small intervals of R contain periodic points of f.

In the scope of this work, chaotic functions are assumed to be functions $f: [0,1] \rightarrow [0,1]$ that exhibit chaotic behavior. Practical implementations of PRNGs based on such chaotic functions (e.g. [1], [9]) have therefore a state s, which comprises a real in the interval $[0,1]$. The PRNG output is computed as a function of the state. Due to the chaotic behavior, it is assumed that chaotic PRNGs have desirable statistical properties. Unfortunately, the restriction to a finite state because of finite number representation can cause problems because the Lyapunov exponent is not greater than zero anymore.

To achieve a reasonably good behavior of the chaotic function, IEEE754 double precision was chosen as numeric representation for the following investigations. A 64-bit state (of which only 62 bits are used to represent [0:1]) is far too large to allow a complete analysis of the state space, so the state space was sampled in order to be able to run an analysis in a reasonable amount of time.

A common and simple chaotic function is the Logistic Map function (cf. [11] and the references therein).

$$f_f: [0,1] \rightarrow [0,1]$$
$$\text{with } f_1(x) = a \cdot x \cdot (1-x), a \approx 4 \quad (1)$$

Table 1 shows that a single component was identified with a comparably low cycle length. According to Section 2, the expected cycle length for a "good" PRNG is $0.78248 \cdot \sqrt{n}$, which would in this case be around $1.68 \cdot 10^9$. The experimentally determined cycle length of $6.62 \cdot 10^6$ is much smaller than the expected value and is therefore not a very good result. The maximum tree height is most probably not representative, as only a small part of the state space was sampled by the analysis. The percentage of start values that belong to the same component is obviously 100%, as only a single component has been detected. The maximum tree height for the 100 start values is about $66 \cdot 10^6$. According to Section 2, the average tree height of a "good" PRNG is expected to be $0.33 \cdot \sqrt{n} \approx 0.71 \cdot 10^9$, so the actual result is a factor of 10 below this.

Table 1. Analysis results of Logistic Map algorithm for 100 start values

Component	Cycle Length	Max Tree Height	Percent Start Values
1	6623920	68244008	100

Another chaotic function is the Trigonometric chaotic function, which was presented in [13].

$$f_2: [0,1] \rightarrow [0,1]$$
$$\text{with } f_2 = \sin^2(z \cdot \arcsin\sqrt{x}), z > 1 \quad (2)$$

Table 2 shows the analysis of the state space for 100 randomly chosen start values. The analysis shows that two components were found. While both components show a better result than the Logistic Map function, the maximum cycle length of $9.7 \cdot 10^6$ is still significantly lower than the theoretical value from Section 2. The maximum tree height for the 100 start values is about $36 \cdot 10^6$, which is a factor of 40 below the expected value.

Thus, similar to the former function, the Trigonometric function is also not well suited for a security relevant PRNG. These observations are in line with earlier investigations, e.g. in [11].

Table 2. Analysis results of Trigonometric Function for 100 start values

Component	Cycle Length	Max Tree Height	Percent Start Values
1	9734369	40443012	33
2	8255730	63779509	64
3	3299245	7632991	3

4. Breaking out of the cycle

A simple way to avoid short cycles is to modify the PRNG algorithm for certain iterations, so that the follow up state is altered compared to the original state transition. If this happens when a cycle has already been entered, and if the follow-up state is chosen randomly, there is a high probability that the new state is located in a tree and not a cycle. The cycle has successfully been broken out of in this case. The high probability of a successful break-out comes from the fact, that the number of cycle nodes is small compared to the number of tree nodes. Figure 3 illustrates what is happening in the break-out case.

In order to keep the overhead small in terms of computational complexity and chip area, the modification to the algorithm that is utilized for

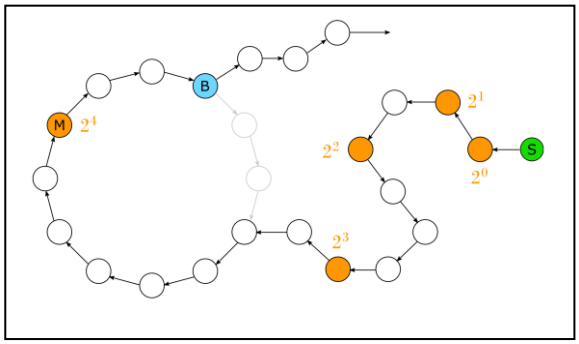

Figure 3. Breaking out of a cycle

breaking out of a cycle should be as small as possible. Also, the decision when to use a different iteration should be simple.

Note that in general, the possible gain in quality is large. If both transition functions were possible and equally likely in each state, then each node in the graph would have two outgoing edges. For randomly chosen edges, our experiments indicate that the graph then is weakly connected, with a strongly connected component that comprises about 84% of the nodes, and a rich inner structure. Let k be the average distance between two break-out states. In this case, k should be chosen small enough that at least one break-out state is on each of the larger cycles.

4.1 Analysis for Logistic Map

Figure 4 shows the analysis results for a break-out to a random node in the state graph for the Logistic Map transition function. As a special case of breaking-out every k steps on average, we are using a counter and break out exactly after k steps, starting with $k = 1$. Experiments have been performed with different values of *k*, ranging from 2 to 1024. The x axis shows the values of *k*, while on the y axis the maximum cycle length that was found for a run with 100 different start values is displayed. The random transition function that was used for the selection of the target node was the AES symmetric stream cipher, used in Cypher Feedback (CFB) mode, meaning that the output of the encryption algorithm was used as input for the next iteration. With AES being a highly regarded cryptographic algorithm, this creates a pseudo random generator of exceptional quality, but even more importantly without any expected statistic dependence on the Logistic Map function. The result shows a very mixed behavior: for some values of *k*, the maximum cycle length is increased compared to the values in Table 2. But e.g. for *k*=64, the maximum cycle length is even lower than without the break-out mechanism. The conclusion that can be drawn is, that a blind break-out

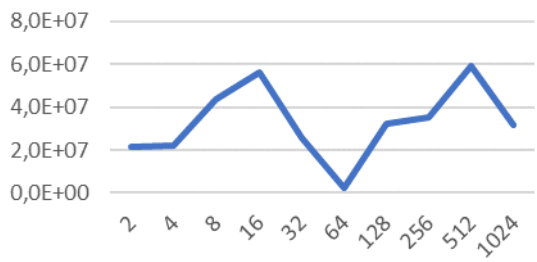

Figure 4. Maximum cycle length for Logistic Map using random break-out target nodes

to a random target node will not generally improve the state space structure, but might randomly improve or degrade it, dependent on the chosen target node.

5. Parameter Modification

A purely random approach to the break-out method apparently does not necessarily have a positive impact, as was shown in the previous section. Furthermore, implementing a statistically independent pseudo random generator in addition to the original transition function does not appear very efficient from a computational point of view, leading to a significantly increased required chip area for hardware implementations. So instead of approximating a random function with expected values as given by [7], we propose to use the chaotic function with a different parametrization instead. The properties of chaotic functions should ensure that the changed parametrization leads to a behavior that is statistically independent enough from the original function.

5.1 Analysis for Logistic Map

Looking at the Logistic Map function $f_1(x) = a \cdot x \cdot (1 - x)$, there is not much potential for parametrization: the only candidate to modify is parameter a.

The analyzed approach was to switch between a value of 3.99 and 3.98 every k steps with k ranging from 2 to 1024. Table 3 shows the result of the state space analysis. It can be seen that breaking out of the cycles increases the maximum cycle length significantly by a minimum factor of 2 for $k = 2$ up to a factor of 227 for $k = 1024$. It can also clearly be seen that the maximum cycle length increases with k. Figure 5 shows this relationship. The graph is not completely consistent with k-values of 32 and 256 showing a decline compared to the previous values.

This can most probably be explained by the very low number of 10 start values for measuring the cycle lengths.

Table 3. Analysis results of Logistic Map Switching to alternative A-parameter every k steps

k	Cycle Lengths	Maximum Tree Heights	% of Start Values
2	14258373	169347594	100
4	81839355	219447222	100
8	92330073	116649348	50
	158592654	153814118	20
	23977917	39516082	20
	5583933	29542408	10
16	166687958	347193629	90
	18456033	66490210	10
32	16463304	509498228	50
	106561026	184708805	50
64	285351755	247720291	70
	27448135	163920805	20
	186333745	138538519	10
128	365089092	459489459	90
	490073193	568127512	10
256	122507017	1011052679	50
	61896137	766210246	50
512	918541890	913649910	70
	505315773	506390293	30
1024	1517222425	2056217674	60
	997359850	1168909136	40

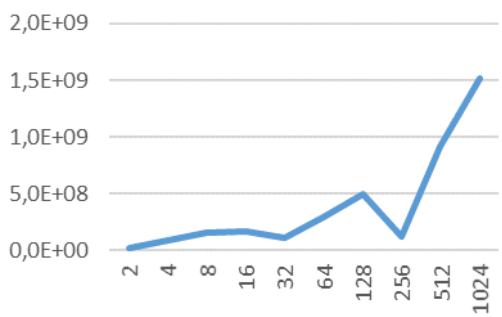

Figure 5. Maximum cycle length over *k* for Logistic Map

5.2 Analysis for Trigonometric Function

Similar investigations have been performed for the Trigonometric function. Again, the function $f_2 = \sin^2(z \cdot \arcsin\sqrt{x})$ has not many options to be parametrized. The most obvious one is a modification of the z parameter. The analyzed modification is a switch between the two z-values 2 and 3 after k iterations with k ranging from 2 to 1024. Table 4 shows the analysis results. Again, the maximum cycle length was increased for all k-values except for $k = 2$ by factors between 7.4 and 75. Figure 5 shows the relationship between *k* and the maximum cycle length and shows a similar dependency as for the Logistic Map function.

Table 4. Analysis results of Trigonometric Function switching to alternative z-parameter every k Steps

k	Cycle Lengths	Maximum Tree Heights	% of Start Values
2	7759233	79946065	90
	7483845	8203387	10
	6666741	51758320	10
4	72851705	101640527	90
	25009350	36265720	10
8	149450121	137849328	90
	18636894	13201151	10
16	132378065	135672835	100
32	380555241	491267040	100
64	60296210	35947884	10
	120161535	89028321	10
	63678680	25751263	10
	191870250	288875294	70
128	182362011	418726507	80
	8997879	244129755	20
256	561225035	231801614	50
	305935113	373691537	50
512	633619125	672871645	40
	218623158	662621246	20
	218623158	508761773	40
1024	728877500	986558422	30
	251601625	529695505	30
	155949650	66719039	40

6. Statistical evaluation

While the positive impact of the break-out method could be demonstrated in the previous section, it seems worthwhile to verify that the change to the transition function does not have a negative impact on its statistical behavior. One of the most commonly used methods to evaluate the statistical properties of pseudo random number generators is the NIST test battery. It comprises 15 different statistical tests that each result in a number of passed and failed test cases. For a "good" PRNG, the proportion of passed to failed test cases is expected to be greater than 96% for each of the tests. Figure 7 shows the number of passed test cases per test, sorted in increasing order for easier comparability. The blue dotted line is the result for the original unmodified Logistic Map and clearly indicates, that using this algorithm as

cryptographic pseudo random number generator without any further modification cannot be recommended due to the significant number of tests with less than 96 passing test cases. The red solid line is the result for the Logistic

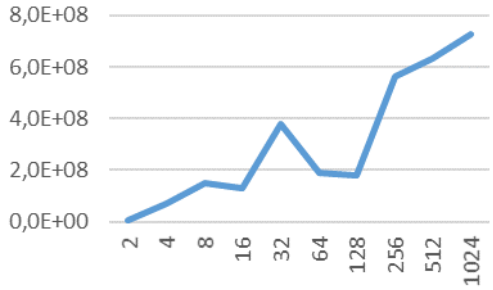

Figure 6. Maximum cycle length over *k* for Trigonometric function

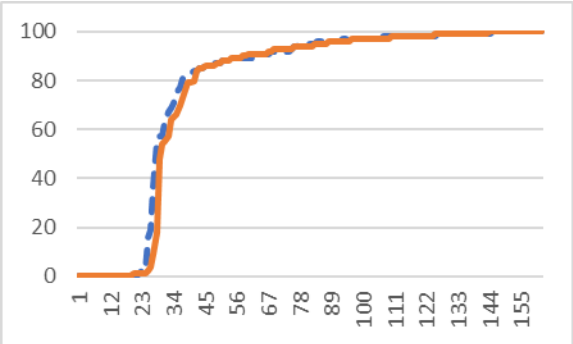

Figure 7. Maximum cycle length over *k* for Trigonometric function

Map using a regular break-out with *k*=1024. The same modified parameterization as presented in Section 5.2 has been used. It can easily be seen that the result of the NIST test suite is comparable to the result for the unmodified algorithm, so the break-out method does not have a negative impact on the statistic behavior of the algorithm in this case.

The same analysis has been performed on the original and modified Trigonometric function, using *k*=1024 and the same parameterization as in Section 5.2. The result shown in Figure 8.depicts, that in this case the result of the NIST test battery is even a bit better for the break-out version of the algorithm than for the original function.

From the analysis above it can be concluded that the break-out method does not affect the statistic properties of the examined chaotic functions in a negative way.

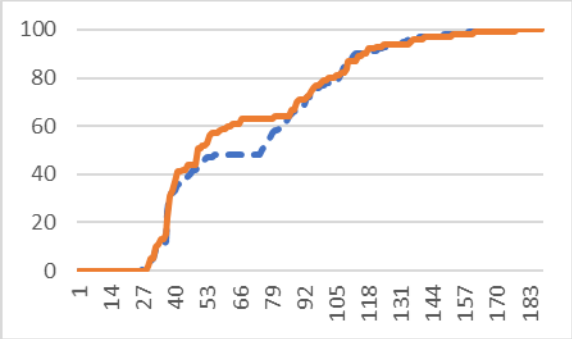

Figure 8. Maximum cycle length over *k* for Trigonometric function

7. Conclusion

A simple method to avoid short cycle lengths in implementations of PRNGs based on chaotic functions was presented. It could be demonstrated that the cycle length for the Logistic Map function can be extended significantly by modifying the parameterization of the chaotic function for certain iterations. This might make chaotic PRNGs usable for an extended range of security applications where increasing the size of the state is not an option because of hardware or computational restrictions. One might even think about this method as a possibility to “repair” an already built-in weak PRNG (even in hardware), as the second transition might be realized in the form of a re-seeding. This approach leads to significant improvements for both investigated chaotic functions (Logistic Map and Trigonometric function).

Further improvements can potentially be achieved by hardcoding an extra transition. In the case of several components, the method of extra transitions could even be extended to link all components together, so that for *all* seed values, a larger cycle length is guaranteed.

Due to the low computational complexity, chaotic algorithms should be investigated further, e.g. in the context of RFID with its limitations on chip area and energy consumption.

As future work, it seems important to gather more statistically relevant data by analyzing a higher number of start values, e.g. on a high-performance computer. Furthermore, extending the investigations towards fix point implementations of chaotic functions, that according to our preliminary experiments seem to exhibit a better behavior than floating point implementations, seems a reasonable way forward.

Finally, the structure of graphs where each node has two outgoing edges might be investigated.

[1] M. Abutaha et al., “Design of a peudo-chaotic number generator as a random number generator”, in International Conference on Communications (COMM), 2016.

[2] A. Beckmann, J. Fedorowicz, J. Keller, and U. Meyer, “A structural analysis of the a5/1 state transition graph,” in First Workshop on GRAPH Inspection and Traversal Engineering, ser. Electronic Proceedings in Theoretical Computer Science, vol. 99. Open Publishing Association, 2012, pp. 5–19.

[3] A. Biryujkov, A. Shamir, and D. Wagner, “Real Time Cryptanalysis of A5/1 on a PC”, in Proceedings of Fast Software Encryption 7th International Workshop, New York, 2000.

[4] A. Desai, A. Hevia, and Y. L. Yin. "A practice-oriented treatment of pseudorandom number generators." International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 2002.

[5] Y. Dodis, D. Pointcheval, S. Ruhault, D. Vergniaud, and D. Wichs, “Security analysis of pseudo-random number generators with input:/dev/random is not robust.” , Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ACM 2013.

[6] K. Entacher, “A collection of selected pseudorandom number generators with linear structures,” ACPC-Austrian Center for Parallel Computation, Tech. Rep., 1997.

[7] P. Flajolet and A. M. Odlyzko, “Random mapping statistics,” in Advances in Cryptology. Springer Verlag, 1990, pp. 329–354.

[8] J. Golic, “Cryptanalysis of Alleged A5 Stream Cypher”, in Proceedings of Advances in Cryptology – Eurocrypt 97, Konstanz, 1997.

[9] M. Hamdi, R. Rhouma, and S. Belghith, "A very efficient pseudo-random number generator based on chaotic maps and S-box tables." Int. J. Comput. Control Quantum Inform. Eng 9 (2015), pp. 481-485.

[10] J. Keller, “Parallel exploration of the structure of random functions,” in Proceedings of the 6th Workshop Parallele Systeme und Algorithmen (PASA) in conjunction with the International Conference on Architecture of Computing Systems, ARCS. VDE, 2002.

[11] J. Keller, H. Wiese, “Period lengths of chaotic pseudo-random number generators.” in Proceedings of the Fourth IASTED International Conference on Communication, Network and Information Security. pp. 7-11. CNIS '07, ACTA Press, Anaheim, CA, USA, 2007, http://dl.acm.org/citation.cfm?id=1659141.1659144, Access Date: 1st October, 2016.

[12] D. E. Knuth, “Mathematical analysis of algorithms.” in Proc. of IFIP Congress 1971, Information Processing 71. pp. 19-27. North-Holland Publ. Co., 1972.

[13] Z. Kotulski, J. Szczepanski, J., K. Górski, A. Górska, A. Paszkiewicz, “On constructive approach to chaotic pseudorandom number generators.” in Proc. Of RCMCIS 2000, Zegrze. pp. 191-203, 2000.

[14] C. Manifavas, G. Hatzivasilis, K. Fysarakis, K. Rantos, “Lightweight cryptography for embedded systems - a comparative analysis.” in Proc. 8th International Workshop on Data Privacy Management and Autonomous Spontaneous Security, pp. 333-349, Springer, 2014, http://dx.doi.org/10.1007/978-3-642-54568-9_21, Access Date: 1st October, 2016.

G. Marsaglia, “The marsaglia random number cdrom including the diehard battery of tests of randomness,” 1995. [Online]. Available:http://www.stat.fsu.edu/pub/ diehard/, Access Date: 1st October, 2016.

[15] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography. CRC Press, 1996.

[16] B. Mennink, B. Preneel, “On the XOR of multiple random permutations.” in Proc. Applied Cryptography and Network Security. pp. 619-634, 2015.

[17] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, “Statistical test suite for random and pseudorandom number generators for cryptographic applications: Special publication 800-22, revision 1a,” 2010. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800022rev1a.pdf, Access Date: 1st October, 2016.

[18] R. Sedgewick, P. Flajolet, “An Introduction to the Analysis of Algorithms.”, Addison-Wesley, Reading Mass., 1996.

[19] J. Szczepanski, Z. Kotulski, “Pseudorandom number generators based on chaotic dynamical systems.”, Open Systems & Information Dynamics 8(2), 137-146 (Jun 2001), http://dx.doi.org/10.1023/A:1011950531970, Access Date: 1st October, 2016.

